DESIGNING AND IMPLEMENTING NETWORK APPLICATION PROTOCOL USING TCP

Course: COMP 3670 – Computer Networks
Instructor: Dr. Sherif Saad

Group 11 Team Members

1. Charles Corro (Did Q2)
2. Keerthana Madhavan (Did Intro, q1.1, q1.2, design diagrams)
3. Van Minh Ngai (Did q1.3, 1.4)
4. Nitin Ramesh (Did q3)

UNIVERSITY OF WINDSOR

Table of Contents

Introduction

The ability to send data and communicate across a group of nodes (end systems) over TCP/IP networks is needed when developing application. In this communication protocol design, we implemented an application using client-server architecture over TCP. TCP is a connection-based protocol that provides our application with a reliable byte-stream channel to allow smooth data flows between end systems. [1] In this a multithreaded client-server architecture, we have a "always on" server that services multiple requests from clients. We used a low-level networking interface method to develop a network application protocol to ensure communication over multiple end systems work properly because of various processes. In this paper, we introduce a simple TCP application using java Socket programming where each jobseeker (client) connects to the job-creator (server), one at a time depending on the server's availability or allow multiple connections for clients with the same task. In this document, the client is the Jobseeker and the server is the Jobcreator.

The Communication Patterns of the Network Application

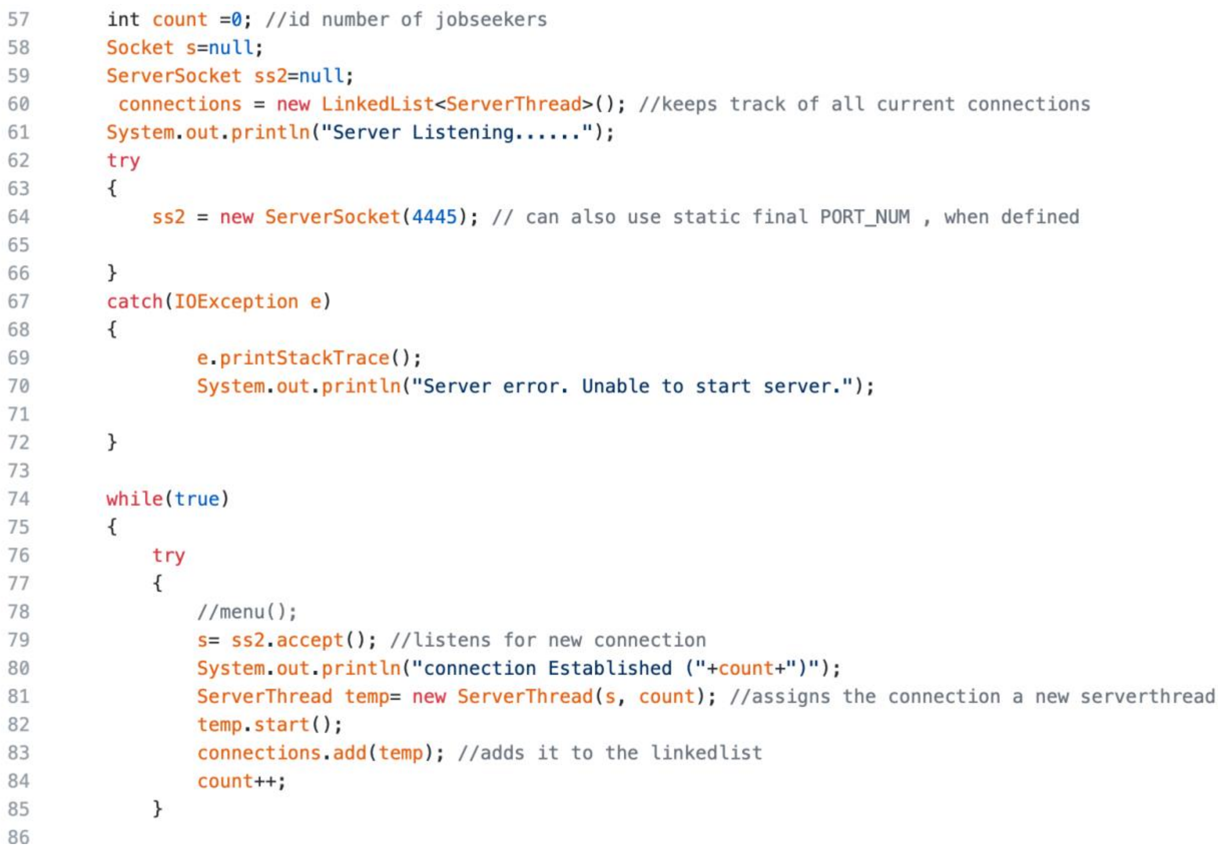
We designed this application using a Client-Server Communication Pattern where one party initiates the communication and the other responds. We use this communication pattern over TCP because of reliable data and large data transmission between two different computers. This protocol can be used by any team of developers to implement a network application. We have decided to split this network application in two nodes: job-creator and jobseeker. On one end of the TCP connection is attached to the job-creator and the other end for the jobseeker. The job-creator node seeks active connection request from a single or multiple jobseeker (clients). Once the connection has been established both the job-creator and jobseeker can send and receive data.

2.1 Job-Creator Overview

The job creator has two important functions that is to start the server and wait for incoming connections over TCP. The job seeker has the following roles: get all the active

[1] The book

connections, assign a job to Job-seeker(s), terminate connections and wait for new connections. The first step for the developer is to set up the socket function to listen for new connections from jobseekers, as you see in the example below.



```java
int count =0; //id number of jobseekers
Socket s=null;
ServerSocket ss2=null;
 connections = new LinkedList<ServerThread>(); //keeps track of all current connections
System.out.println("Server Listening......");
try
{
    ss2 = new ServerSocket(4445); // can also use static final PORT_NUM , when defined

}
catch(IOException e)
{
        e.printStackTrace();
        System.out.println("Server error. Unable to start server.");

}

while(true)
{
    try
    {
        //menu();
        s= ss2.accept(); //listens for new connection
        System.out.println("connection Established ("+count+")");
        ServerThread temp= new ServerThread(s, count); //assigns the connection a new serverthread
        temp.start();
        connections.add(temp); //adds it to the linkedlist
        count++;
    }

```

The socket in job-creator can be closed after accepting a connection or left open for multiple connections. Since our application is designed to accept multiple jobseekers, we assign incoming connections to a LinkedList called server-thread and monitor for active/inactive connections on the server.

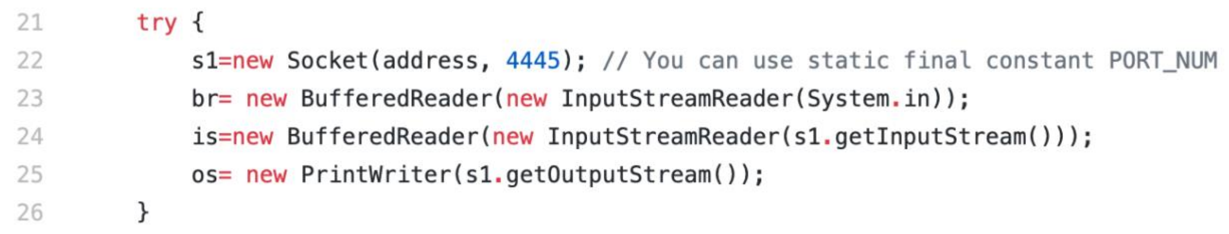
Assigning a job to Jobseekers

The job creator main responsibility to monitor and assign jobs to job-seekers but it should satisfy the following conditions: check for availability of the job-seeker and assign the job, ask the client which job they should be assigned based on pre-defined set of jobs (our design will have 2 jobs), assign multiple jobs to the seeker, but only one job at a time, terminate connections when necessary and most importantly perform error handling.

2.2 Job-Seeker Overview

The jobseeker has a job to initiate contact with the job-creator when the server is up and running. This is done by the job-seeker TCP socket that specifies the address of the job creator socket, IP address and the port number.

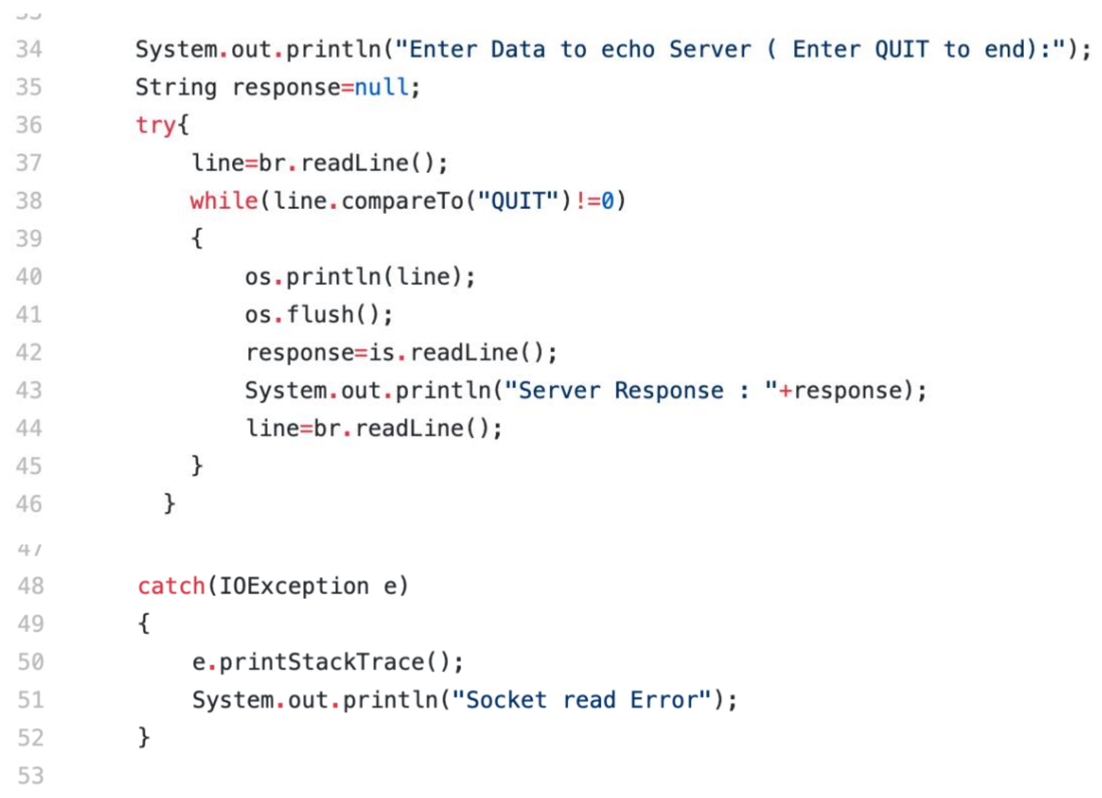
TCP Job-Seeker Client: Establish Connection to Job-Creator



```java
try {
    s1=new Socket(address, 4445); // You can use static final constant PORT_NUM
    br= new BufferedReader(new InputStreamReader(System.in));
    is=new BufferedReader(new InputStreamReader(s1.getInputStream()));
    os= new PrintWriter(s1.getOutputStream());
}
```

After creating the socket, the client initiates a handshake with the job creator and establishes a TCP connection. Now when sending a message, the client process can send bytes into its socket and the server will receive the in the same order sent by the job seeker. The server will then send bytes back to the client to notify whether the input was accepted or not.

TCP Job-Seeker Client: Send Message and Receive Message

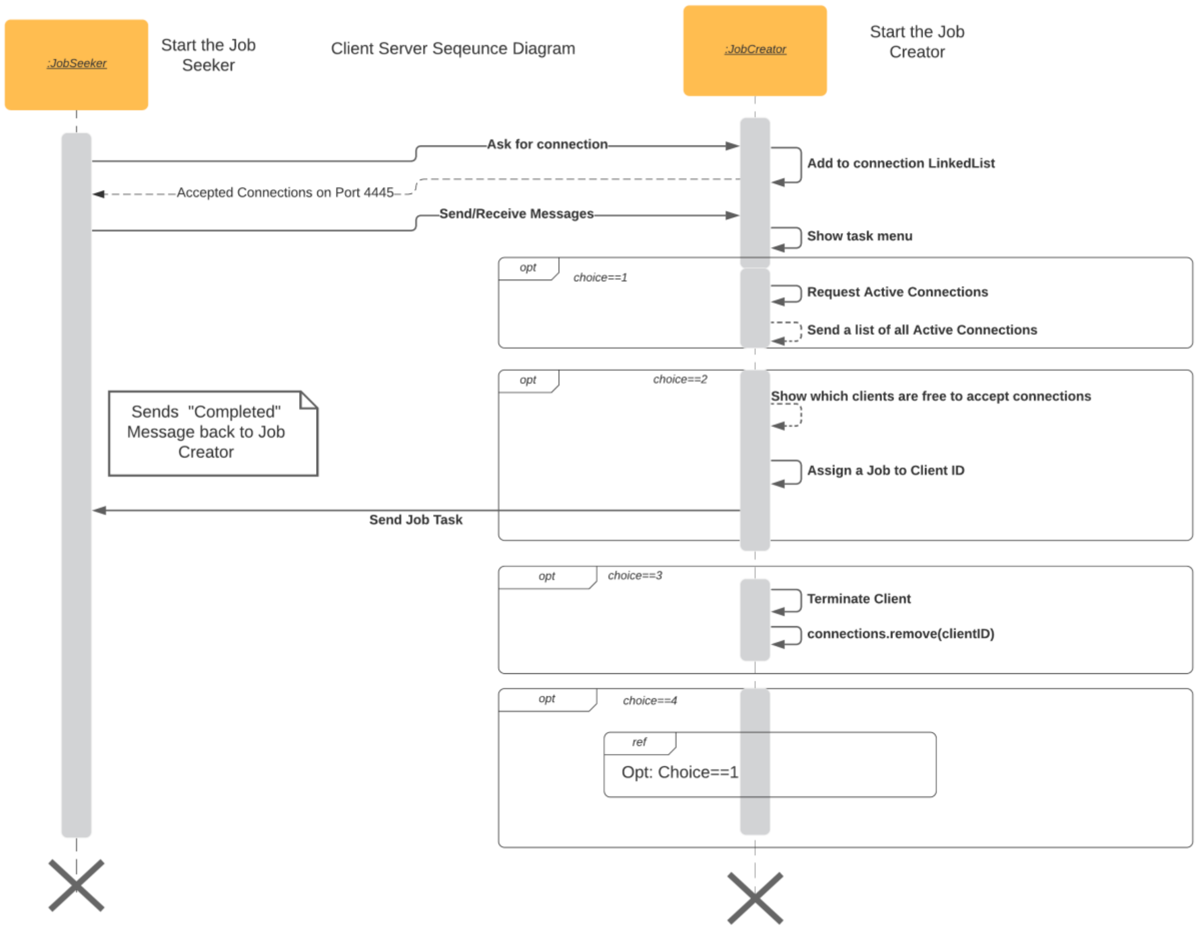


```java
System.out.println("Enter Data to echo Server ( Enter QUIT to end):");
String response=null;
try{
    line=br.readLine();
    while(line.compareTo("QUIT")!=0)
    {
        os.println(line);
        os.flush();
        response=is.readLine();
        System.out.println("Server Response : "+response);
        line=br.readLine();
    }
  }

catch(IOException e)
{
    e.printStackTrace();
    System.out.println("Socket read Error");
}
```

Completing Given Tasks

The job seeker main responsibility is to accept or reject a job from the server, complete the tasks and report job status and results to the creator. Additionally, the jobseeker can choose to terminate any time during active connection session.

Sequence Diagram


:JobSeeker
Start the Job Seeker
Client Server Seqeunce Diagram
:JobCreator
Start the Job Creator
Ask for connection
Add to connection LinkedList
Accepted Connections on Port 4445
Send/Receive Messages
Show task menu
opt
choice==1
Request Active Connections
Send a list of all Active Connections
opt
choice==2
Show which clients are free to accept connections
Sends "Completed" Message back to Job Creator
Assign a Job to Client ID
Send Job Task
opt
choice==3
Terminate Client
connections.remove(clientID)
opt
choice==4
ref
Opt: Choice==1

Protocol Design Goals

Our network application protocol framework should be simple, fast, must ensure reliable exchanges, accept multiple job seekers, and provide error handling for socket connections, and send/receive messages both ways. Our communication protocol should be simple and must use a TCP connection because TCP is a standard network protocol and used for reliable data transmission. We don't want to make easy tasks hard or have multiple ways to accomplish a common goal of sending and receiving messages over a TCP connection. Our protocol should also be scalable, we designed the protocol so that it balances the job-creator responsibilities of communicating with single or multiple jobseekers, and also implement a mechanism to free the servers if there is inactive connections. Our protocol should also be efficient, to ensure this, our application is designed to make us of java collections like LinkedLists to create new instances of ServerThread Class that processes the client messages for new connections. This ensures that we don't create a message mechanism separately for each incoming connection, but rather create instances of a base "message mechanism" class ServerThread. Finally, our network application protocol should be designed to add features or designed to make further extensions in future.

Define the message format, structure, and semantics

The message format is the fundamental of the network application protocol. This message format should be formative and short. In our code, we made each message as a String object with either "done5" or "done10" as the instruction. We believe this is a better way to implement this message format into the protocol because we can change it in the future without having to change a major part of our code. Each message will be handle differently based on their value (String). As for the message structure, we coded the program so we can track and end the message. Track is the priority for the program to begin the tracking message, after that the program can end the message or can send the information to a target.

Design the communication Rules

The communication rules for this network application protocol is a multithreaded approach. First, for the sending messages from and to the server, a client must connect to the server, so it can be assigned a new thread for an uninterrupted line of communication, on the server end. Then, the client is ready to be assigned a job by the server. When the server chooses said client for a job, a command string is chosen to be sent to the client, to trigger it to do a task. After that, the message is officially sent over the network, through port 4445. On receiving the instruction from the server, the client begins its job and after it is done. It sends back a string confirming the completion of the task, informing the server that it is now available for a new job. To stop communication:

(i) The server can terminate connection with a single client by using the menu on the JobCreator terminal. The client will not be able to communicate with the server anymore. (Implemented to prevent malicious clients)

(ii) The client can terminate the connection by entering ctrl-c on their terminal. The server will not immediately know that the client has terminated the connection. It will be known only when the server decides to assign a job to said client.

Argue the need for New Application Layer Protocol

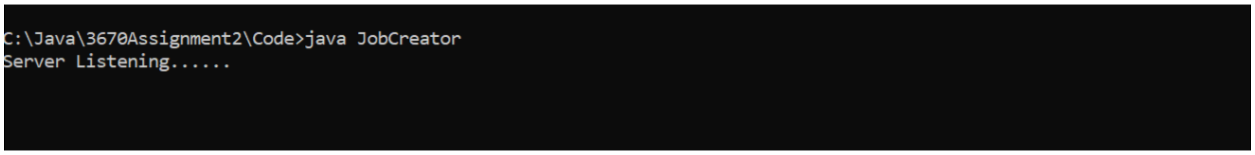
For our network application, a new application layer protocol is necessary instead of using existing standard protocols like HTTP, SMTP, WebSocket, etc. The reason for this is because due to many reasons. First unlike HTTP or WebSocket, our application will have multiple lasting connections to various clients from the server. When connected the server will have a priority list of jobs sent by those clients. Also, since the application does not send emails with addresses, then there is no need for the use of the SMTP protocol. Since this application is will send tasks and the way the connections are formatted are going to be different, we will need to implement a new application layer protocol. We chose a new TCP application protocol because of the reliable data transmission over clients and the server.

GitHub Repository to Code

https://github.com/NitinR99/3670Assignment2

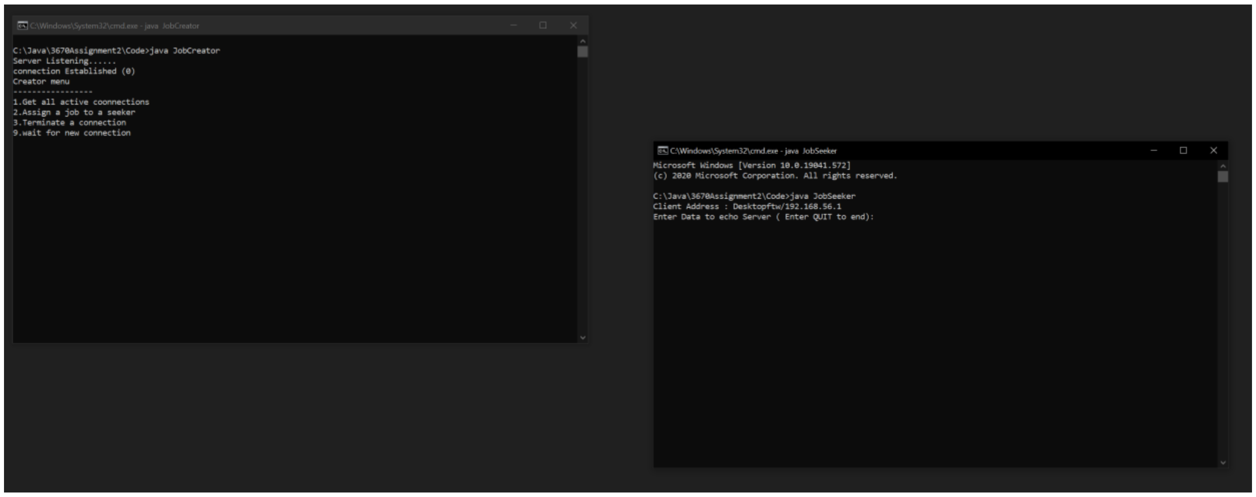
Screenshot

Step 1: Screenshot



```
C:\Java\3670Assignment2\Code>java JobCreator
Server Listening......
```

Step 2: Run JobSeeker in a new command prompt windows

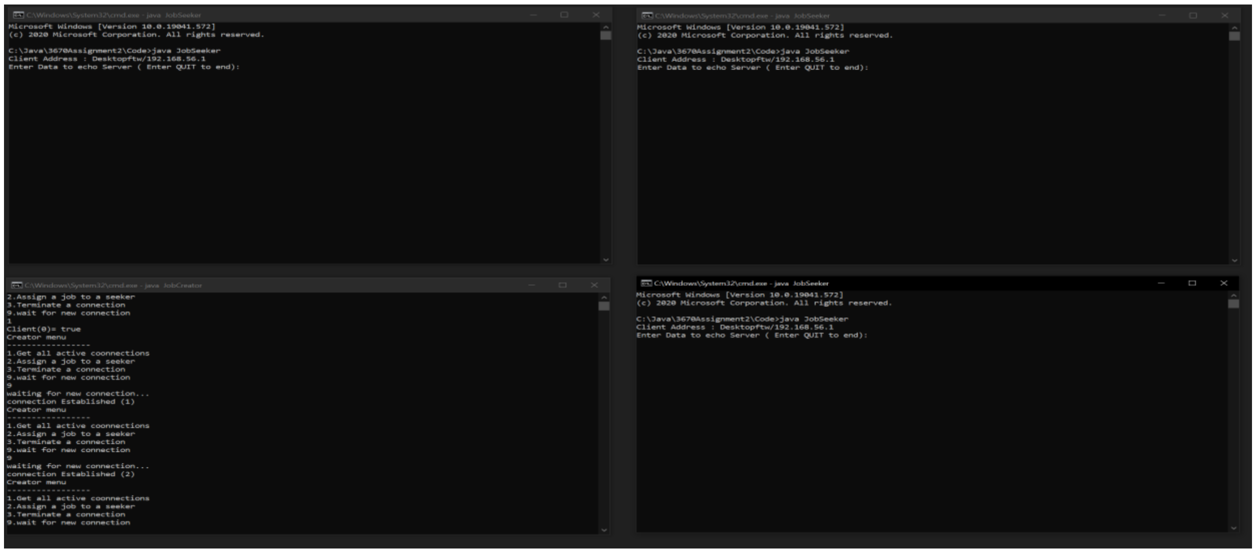


C:\Windows\System32\cmd.exe - java JobCreator
C:\Windows\System32\cmd.exe - java JobSeeker

```
C:\Java\3670Assignment2\Code>java JobCreator
Server Listening......
connection Established (0)
Creator menu
----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
```

```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
```

Step 3: You can also run multiple JobSeekers to connect to the same JobCreator, or you can skip this step.



C:\Windows\System32\cmd.exe - java JobSeeker
C:\Windows\System32\cmd.exe - java JobSeeker
C:\Windows\System32\cmd.exe - java JobCreator
C:\Windows\System32\cmd.exe - java JobSeeker

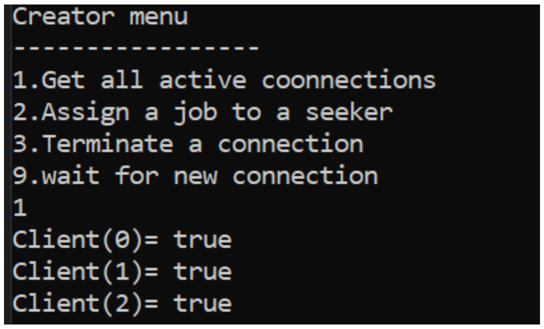
```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
```

```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
```

```
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
1
Client(0)= true
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
9
waiting for new connection...
connection Established (1)
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
9
waiting for new connection...
connection Established (2)
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
```

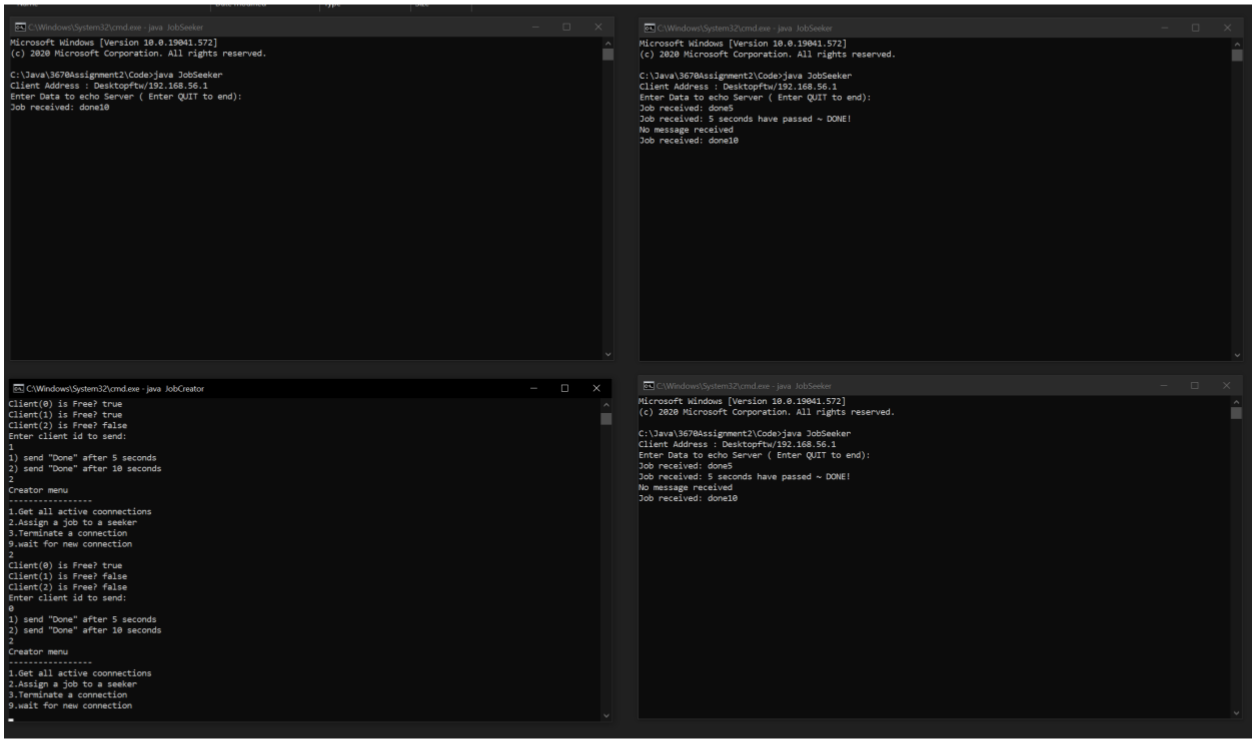
```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
```

Step 4: You can view the current connections to the JobCreator by choosing option “1” on the JobCreator command window.



```
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
1
Client(0)= true
Client(1)= true
Client(2)= true
```

Step 5: Now, you can assign jobs to any connect JobSeeker by selecting “option 2” and selecting a free client.



C:\Windows\System32\cmd.exe - java JobSeeker

```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
Job received: done10
```

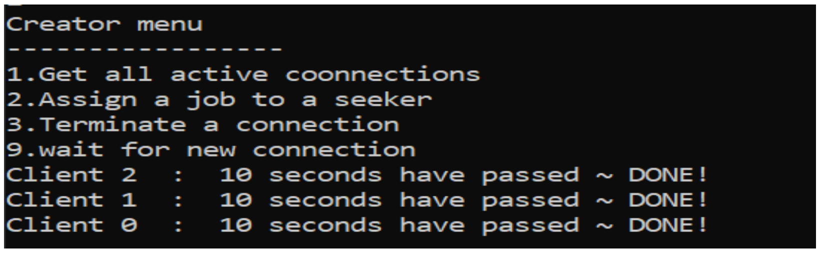
C:\Windows\System32\cmd.exe - java JobSeeker

```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
Job received: done5
Job received: 5 seconds have passed ~ DONE!
No message received
Job received: done10
```

```
Client(0) is Free? true
Client(1) is Free? true
Client(2) is Free? false
Enter client id to send:
1
1) send "Done" after 5 seconds
2) send "Done" after 10 seconds
2
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
2
Client(0) is Free? true
Client(1) is Free? false
Client(2) is Free? false
Enter client id to send:
0
1) send "Done" after 5 seconds
2) send "Done" after 10 seconds
2
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
```

```
Microsoft Windows [Version 10.0.19041.572]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Java\3670Assignment2\Code>java JobSeeker
Client Address : Desktopftw/192.168.56.1
Enter Data to echo Server ( Enter QUIT to end):
Job received: done5
Job received: 5 seconds have passed ~ DONE!
No message received
Job received: done10
```

After any client is done with its assigned job, it will send a confirmation string to the JobCreator to be displayed.



```
Creator menu
-----------------
1.Get all active coonnections
2.Assign a job to a seeker
3.Terminate a connection
9.wait for new connection
Client 2  :  10 seconds have passed ~ DONE!
Client 1  :  10 seconds have passed ~ DONE!
Client 0  :  10 seconds have passed ~ DONE!
```

Additional Information

At any time during the execution, the JobSeeker can terminate connection by entering ctrl-c on their command window. The JobCreator is also able to terminate any unwanted connection by using menu option "3".

Appendix

JobSeeker.java

```
import
java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;
/*
This class is the server which is able to connect to multiple
JobSeekers (clients)
*/
public class JobCreator {
   static LinkedList<ServerThread> connections; //each node in
the linkedlist contains the thread of a jobseeker. Allowing this
code to assign jobs to multiple clients simultaneously
   static void menu() throws Exception
   {
    BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Creator menu\n-----------------\n1.Get
all active coonnections\n2.Assign a job to a seeker\n3.Terminate
a connection\n9.wait for new connection");
    int choice=-1;
    try{
    choice=Integer.parseInt(br.readLine());} //reads choice the
user selects
    catch(Exception e)
    {
        System.out.println("input error.");
    }
    if(choice==1)//lists all active connections
    {
        for(int i=0;i<connections.size();i++)
        {
            System.out.println("Client("+i+")=
"+connections.get(i).isActive);
        }
        menu();
    }
    else if(choice==2)//assigns job to a client
    {
              //shows user the clients which are free and can
accept a job
              for(int i=0;i<connections.size();i++)
              {
                     System.out.println("Client("+i+") is Free?
"+connections.get(i).isFree);
              }

              System.out.println("Enter client id to send:
");//gets the client id to assign a job to it
        int clId=Integer.parseInt(br.readLine());
              if(connections.get(clId)!=null &&
connections.get(clId).isFree)
        {
            try{//Asks user which of the 2 jobs should be
assigned to the client
                            System.out.println("1) send \"Done\"
after 5 seconds");
                            System.out.println("2) send \"Done\"
after 10 seconds");

                            int choi =
Integer.parseInt(br.readLine());

                            if(choi == 1){
                                   String t = "done5";

       connections.get(clId).os.println(t); //sending task prompt

       connections.get(clId).os.flush();
                            }
                            else if(choi == 2){
                                   String t = "done10";

       connections.get(clId).os.println(t); //sending task prompt
```

```
        connections.get(clId).os.flush();
                                }
                                connections.get(clId).isFree =
false;

                }
                catch(Exception e)
                {
                    System.out.println("error while sending the
task");
                }
                menu();
            }
            else if(connections.get(clId)!=null &&
!connections.get(clId).isFree)
            {
                    System.out.println("The selected client is not
free. Try again.");
                    menu();
            }
            else{
                System.out.println("Client not found. Try again.");
                menu();
            }
        }
        else if(choice==3)//Terminates the server side connection of
the client. The client will not be able to send anything to the
server.
        {
            System.out.println("Enter client id to terminate: ");
            int clId=Integer.parseInt(br.readLine());
            if(connections.get(clId)!=null)
            {
                try{
                connections.get(clId).is.close();
                connections.get(clId).os.close();
                connections.get(clId).s.close();
                connections.get(clId).isActive=false;
                connections.remove(clId);}
                catch(Exception e)
                {
```

```
                System.out.println("error while ending client
connection");
                menu();
            }
        }
        menu();
    }
    else if(choice==9)
    {
        System.out.println("waiting for new connection...");
    }
    else{
        System.out.println("Invalid option. Try again.");
       menu();
    }

   }
public static void main(String args[]) throws Exception{

    int count =0; //id number of jobseekers
    Socket s=null;
    ServerSocket ss2=null;
     connections = new LinkedList<ServerThread>(); //keeps track
of all current connections
    System.out.println("Server Listening......");
    try
    {
        ss2 = new ServerSocket(4445); // connection is through
port 4445

    }
    catch(IOException e)
    {
            e.printStackTrace();
            System.out.println("Server error. Unable to start
server.");

    }

    while(true)
    {
        try
       {
            s= ss2.accept(); //listens for new connection
            System.out.println("connection Established
("+count+")");
            ServerThread temp= new ServerThread(s, count);
//assigns the connection a new serverthread
            temp.start();
            connections.add(temp); //adds it to the linkedlist
            count++;
        }

    catch(Exception e)
    {
        e.printStackTrace();
        System.out.println("Connection Error. Unable to connect
to Job seekers.");

    }


    for(int i=0;i<connections.size();i++) //checks and removes
inactive connections
    {
        if(connections.get(i).isActive==false)
        {
            connections.remove(i);
            System.out.println("removed a connection");
        }
    }
    menu();

    }

}

}
/*
This class defines the individual thread for each connection to
the JobCreator
 */
class ServerThread extends Thread{
    int id; //unique id for the client, to be used by JobCreator
    boolean isActive = true; // boolean value to check if the
connection is still active
```

```
        boolean isFree = true; // boolean value to check if the
client is doing a task or whether it is free
    String line = null;
    BufferedReader  is = null;
    PrintWriter os = null;
    Socket s = null;
    String fromClient = null;
    public ServerThread(Socket s,int x){
        this.s = s;
        id = x;
    }

    public void run() {
    try{
        is=new BufferedReader(new
InputStreamReader(s.getInputStream()));
        os=new PrintWriter(s.getOutputStream());

    }catch(IOException e){
        System.out.println("IO error in server thread");
    }
//to assign boolean value to isFree
    try {
        line=is.readLine();
        while(line.compareTo("QUIT")!=0){

            os.println(line);
            os.flush();
            System.out.println("Client "+id+"  :  "+line);
                      if(line.equals("5 seconds have passed ~
DONE!")){
                              isFree = true;
                      }
                      if(line.equals("10 seconds have passed ~
DONE!")){
                              isFree = true;
                      }


            line=is.readLine();
        }
    } catch (IOException e) {
        line=this.getName(); //reused String line for getting
thread name
        System.out.println("IO Error/ Client ("+ id+ ")
terminated abruptly");
        isActive=false; //since there is a connection error
    }
    catch(NullPointerException e){
        line=this.getName(); //reused String line for getting
thread name
        System.out.println("Client id("+ id+") Closed");
        isActive=false; //since there is a connection error
    }
//to close the connection
    finally{
    try{
        if (is!=null){
            is.close();
            isActive=false;
        }

        if(os!=null){
            os.close();
            isActive=false;
        }
        if (s!=null){
        s.close();
           isActive=false;
        }

        }
    catch(IOException ie){
        System.out.println("Socket Close Error");
    }
    }//end finally
    }
}
```

JobCreator.java

```
import
java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.LinkedList;
/*
This class is the server which is able to connect to multiple
JobSeekers (clients)
*/
public class JobCreator {
   static LinkedList<ServerThread> connections; //each node in
the linkedlist contains the thread of a jobseeker. Allowing this
code to assign jobs to multiple clients simultaneously
   static void menu() throws Exception
   {
    BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
    System.out.println("Creator menu\n-----------------\n1.Get
all active coonnections\n2.Assign a job to a seeker\n3.Terminate
a connection\n9.wait for new connection");
    int choice=-1;
    try{
    choice=Integer.parseInt(br.readLine());} //reads choice the
user selects
    catch(Exception e)
    {
        System.out.println("input error.");
    }
    if(choice==1)//lists all active connections
    {
        for(int i=0;i<connections.size();i++)
        {
            System.out.println("Client("+i+")=
"+connections.get(i).isActive);
        }
        menu();
    }
    else if(choice==2)//assigns job to a client
    {
              //shows user the clients which are free and can
accept a job
              for(int i=0;i<connections.size();i++)
              {
                     System.out.println("Client("+i+") is Free?
"+connections.get(i).isFree);
              }

              System.out.println("Enter client id to send:
");//gets the client id to assign a job to it
        int clId=Integer.parseInt(br.readLine());
              if(connections.get(clId)!=null &&
connections.get(clId).isFree)
        {
            try{//Asks user which of the 2 jobs should be
assigned to the client
                            System.out.println("1) send \"Done\"
after 5 seconds");
                            System.out.println("2) send \"Done\"
after 10 seconds");

                            int choi =
Integer.parseInt(br.readLine());

                            if(choi == 1){
                                   String t = "done5";

       connections.get(clId).os.println(t); //sending task prompt

       connections.get(clId).os.flush();
                            }
                            else if(choi == 2){
                                   String t = "done10";

       connections.get(clId).os.println(t); //sending task prompt
```

```
        connections.get(clId).os.flush();
                              }
                              connections.get(clId).isFree =
false;

            }
            catch(Exception e)
            {
                System.out.println("error while sending the
task");
            }
            menu();
        }
        else if(connections.get(clId)!=null &&
!connections.get(clId).isFree)
        {
                System.out.println("The selected client is not
free. Try again.");
                menu();
        }
        else{
            System.out.println("Client not found. Try again.");
            menu();
        }
    }
    else if(choice==3)//Terminates the server side connection of
the client. The client will not be able to send anything to the
server.
    {
        System.out.println("Enter client id to terminate: ");
        int clId=Integer.parseInt(br.readLine());
        if(connections.get(clId)!=null)
        {
            try{
            connections.get(clId).is.close();
            connections.get(clId).os.close();
            connections.get(clId).s.close();
            connections.get(clId).isActive=false;
            connections.remove(clId);}
            catch(Exception e)
            {
                System.out.println("error while ending client
connection");
                menu();
            }
        }
        menu();
    }
    else if(choice==9)
    {
        System.out.println("waiting for new connection...");
    }
    else{
        System.out.println("Invalid option. Try again.");
       menu();
    }

  }
public static void main(String args[]) throws Exception{

    int count =0; //id number of jobseekers
    Socket s=null;
    ServerSocket ss2=null;
     connections = new LinkedList<ServerThread>(); //keeps track
of all current connections
    System.out.println("Server Listening......");
    try
    {
        ss2 = new ServerSocket(4445); // connection is through
port 4445

    }
    catch(IOException e)
    {
            e.printStackTrace();
            System.out.println("Server error. Unable to start
server.");

    }

    while(true)
    {
        try
       {
            s= ss2.accept(); //listens for new connection
            System.out.println("connection Established
("+count+")");
            ServerThread temp= new ServerThread(s, count);
//assigns the connection a new serverthread
            temp.start();
            connections.add(temp); //adds it to the linkedlist
            count++;
        }

    catch(Exception e)
    {
        e.printStackTrace();
        System.out.println("Connection Error. Unable to connect
to Job seekers.");

    }


    for(int i=0;i<connections.size();i++) //checks and removes
inactive connections
    {
        if(connections.get(i).isActive==false)
        {
            connections.remove(i);
            System.out.println("removed a connection");
        }
    }
    menu();

    }

}

}
/*
This class defines the individual thread for each connection to
the JobCreator
 */
class ServerThread extends Thread{
    int id; //unique id for the client, to be used by JobCreator
    boolean isActive = true; // boolean value to check if the
connection is still active
```

```
        boolean isFree = true; // boolean value to check if the
client is doing a task or whether it is free
    String line = null;
    BufferedReader  is = null;
    PrintWriter os = null;
    Socket s = null;
    String fromClient = null;
    public ServerThread(Socket s,int x){
        this.s = s;
        id = x;
    }

    public void run() {
    try{
       is=new BufferedReader(new
InputStreamReader(s.getInputStream()));
        os=new PrintWriter(s.getOutputStream());

    }catch(IOException e){
        System.out.println("IO error in server thread");
    }
//to assign boolean value to isFree
    try {
        line=is.readLine();
        while(line.compareTo("QUIT")!=0){

            os.println(line);
            os.flush();
            System.out.println("Client "+id+"  :  "+line);
                      if(line.equals("5 seconds have passed ~
DONE!")){
                              isFree = true;
                      }
                      if(line.equals("10 seconds have passed ~
DONE!")){
                              isFree = true;
                      }


            line=is.readLine();
        }
    } catch (IOException e) {
        line=this.getName(); //reused String line for getting
thread name
        System.out.println("IO Error/ Client ("+ id+ ")
terminated abruptly");
        isActive=false; //since there is a connection error
    }
    catch(NullPointerException e){
        line=this.getName(); //reused String line for getting
thread name
        System.out.println("Client id("+ id+") Closed");
        isActive=false; //since there is a connection error
    }
//to close the connection
    finally{
    try{
        if (is!=null){
            is.close();
            isActive=false;
        }

        if(os!=null){
            os.close();
            isActive=false;
        }
        if (s!=null){
        s.close();
           isActive=false;
        }

        }
    catch(IOException ie){
        System.out.println("Socket Close Error");
    }
    }//end finally
    }
}
```